ERB-957

NRCC No. 24081

National Research Council Canada

Conseil national de recherches Canada

Division of Electrical Engineering

Division de génie électrique

INVESTIGATION OF LEARNING BEHAVIOUR OF A CMAC CELL

P. CHARTIER

DECEMBER 1984

Canada

85-3215/

Copyright 1984 by
National Research Council of Canada

Permission is granted to quote short excerpts and to reproduce figures and tables from this report, provided that the source of such material is fully acknowledged.

Copyright 1984 par
Conseil national de recherches du Canada

Il est permis de citer de courts extraits et de reproduire des figures ou tableaux du présent rapport, à condition d'en identifier clairement la source.

Additional copies are available free of charge from:

Editorial Office, Room 201
Division of Electrical Engineering
National Research Council of Canada,
Ottawa, Ontario, Canada
K1A 0R6

Des exemplaires supplémentaires peuvent être obtenus gratuitement à l'adresse suivante:

Bureau de rédaction, pièce 201
Division de génie électrique
Conseil national de recherches du Canada
Ottawa (Ontario)
K1A 0R6

INVESTIGATION OF LEARNING BEHAVIOUR OF A CMAC CELL

P. Chartier

ERB-957
NRCC No. 24091

DECEMBER 1984

TABLE OF CONTENTS

	Page
ABSTRACT/RESUME	v
INTRODUCTION	1
THE STRUCTURE OF THE CMAC	1
Global Structure of the CMAC in C	2
CYCLCMAC.C	4
THE EXPERIMENTS	5
Experiment 1	7
Experiment 2	9
Experiment 3	12
Experiment 4	14
GENERAL ASSESSMENT	17
Possible Applications	17
Areas for Future Investigation	17
ACKNOWLEDGEMENTS	18
REFERENCES	18
APPENDIX A	19
stdeclar.c	19
consulte.c	21
cyclcmac.c	22

LIST OF FIGURES

1.	CMAC Model	2
2.	The S→M→A→p Mapping	3
3.	The evaluation of quantization levels	4
4.	Dual CMAC arrangement used in the experiments	6
5.	Learning of information (simple waveform)	8
6.	Experiment 2	9
7.	Tracking effect	10
8.	Experiment 2	11
9.	Gain vs. number of learning loops	13
10.	Gain vs. number of learning loops	14
11.	Gain vs. number of learning loops	15
12.	Learning of a complex path	16

ABSTRACT

The CMAC (Cerebellar Modeling Arithmetic Computer) has been implemented on a DEC-VAX computer as a C language subroutine to investigate its properties of generalization and cross-talk, relative to various values of its operational parameters.

We describe the CMAC, its C implementation, and the experiments that have been conducted to evaluate its properties. The program allows all the parameters of a CMAC to be modified, and new cell clusters to be created at run time.

RÉSUMÉ

Le CMAC (Ordinateur arithmétique de modélisation du cerveau) a été programmé sur ordinateur DEC-VAX sous forme d'une fonction en langage "C" dans le but d'examiner ses propriétés de généralisation et d'interférence, pour diverses valeurs de ses paramètres opérationnels.

On décrit ici le CMAC, sa représentation en langage "C" et les expériences qui tendent à l'évaluation de ses propriétés. Il est possible de modifier dans ce programme tous les paramètres du CMAC et de créer en cours d'exécution de nouveaux amas de cellules.

INVESTIGATION OF LEARNING BEHAVIOUR OF A CMAC CELL

P. Chartier

INTRODUCTION

The CMAC (Cerebellar Modeling Arithmetic Computer), from Albus (1), is a simple algorithm modeling a cluster of cerebellar nerve cells. The device constitutes a type of Perceptron (2) and has all the attributes of a classifier. The possibilities it offers in quantifying the membership to a given class makes it an interesting investigation tool for research on fuzzy sets.

This report will concentrate on the implementation of an automaton based on the CMAC, its training and subsequent operation in various conditions of initial state, learned pattern complexity and terminal state, after describing the CMAC's structure and its C implementation.

THE STRUCTURE OF THE CMAC

The CMAC is fully characterized by the following parameters:

$S_1, S_2, \ldots, S_n$: stimuli inputs
$\hat{p}$: training set input
p: response output
Q: quantization factor
j: learning gain
IL: cardinality of the set of input values

and by the following mapping

S→M→A→p

where S→M models the stimuli transmission by mossy fibers, M→A represents connexions from mossy fibers to granule cells, and A→p represents nerve interaction from the granule cells to the Purkijne output cells (Fig. 1).

The value of the output p is determined by individual weights of A→p mapping. Those weight tables are especially important in that they are the repository of a CMAC's knowledge and model an important aspect of low level memory.

Further details are available in the original book by Albus (1). The scope of the C implementation of the CMAC was to faithfully reproduce that

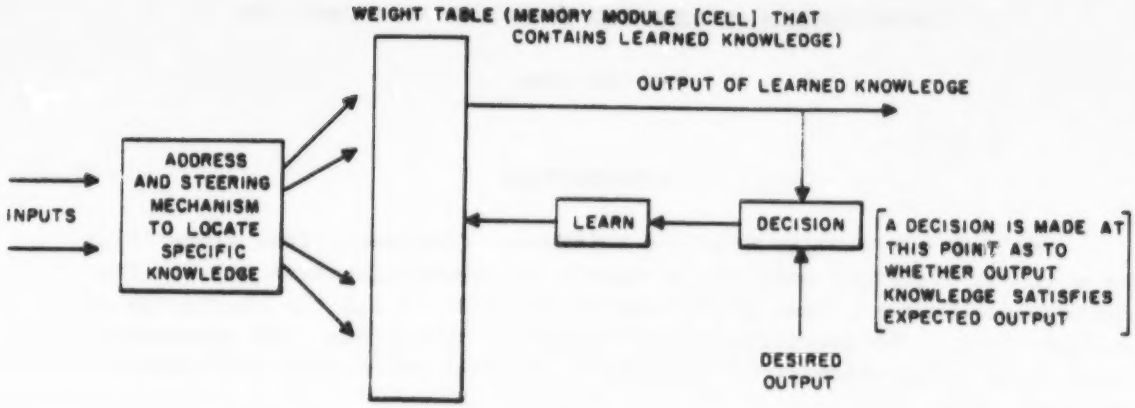
WEIGHT TABLE (MEMORY MODULE [CELL] THAT CONTAINS LEARNED KNOWLEDGE)
OUTPUT OF LEARNED KNOWLEDGE
INPUTS
ADDRESS AND STEERING MECHANISM TO LOCATE SPECIFIC KNOWLEDGE
LEARN
DECISION
A DECISION IS MADE AT THIS POINT AS TO WHETHER OUTPUT KNOWLEDGE SATISFIES EXPECTED OUTPUT
DESIRED OUTPUT

Figure 1. CMAC Model

model as given by Albus; the program has been made into a subroutine receiving as formal arguments the operating parameters of the original CMAC.

Global Structure of the CMAC in C

From now on, the acronym CMAC will exclusively refer to the subroutine implementing Albus's algorithm.

The CMAC has been completely written as reentrant code as a function which dynamically allocates the working storage it needs. This practice implies that in a simulation, every reference to CMAC represents an independant cluster of cells. There are plans, in fact, for a model that would reproduce cells ad lib.

In fact, the actual cluster is represented by a data structure which contains every parameter the CMAC needs to function. The arguments passed to CMAC are used to update that data base. In this fashion, a new independant CMAC can be brought into existence at every invocation of the subroutine.

The pivot function is CONSULTE. It is used to read the weight tables according to the S→M→A→p mapping.

We denote by L the set of all possible values of s_i. The cardinality of L is used along with Q, the quantization factor, to calculate the size of the smallest array that will contain one quantization plane. That size is called DIMTAB. Therefore, the memory allocated to CMAC is stored as Q planes, each plane being a square array of [DIMTAB * DIMTAB] elements, named TABPON.

Figure 2 has been extracted from CONSULTE; it represents the code which performs the weight selection and the integration over the Q plane.

It can be seen that the mapping is represented by a basic array addressing mechanism.

A CMAC can be created, programmed, interrogated respectively by CREECMAC, CONSULTE, CYCLCMAC, and TUECMAC (the listings are in APPENDIX A). In fact CONSULTE is called by CYCLCMAC and CYCLCMAC has a parameter telling it whether to LEARN the value at p or not.

A number of other functions, beyond the scope of this report, have also been developed to preset a CMAC's weight tables with some initial values from a file, to save them, or to gather statistics on the contents of those memory locations.

Finally, the experiments described in the rest of this report were supported by various interactive programs each invoking the aforementioned code.

In summary, our implementations of S→M are procedural, as is M→A. A→p is both tabular and procedural. This has the consequence that CMAC's S→M

SELECTION

```
p = 0.0;

for (i = 0; i < cmac->q; ++i)
                    /* on utilise tabpon comme un tableau tridimensionnel */

   p += cmac->tabpon [
      (( s1 + i ) / cmac->q ) * cmac->dimtab * cmac->q + /* indice s1 (x)*/
      (( s2 + i ) / cmac->q ) * cmac->q +                /* indice s2 (y)*/
      i ] ;                                          /* indice tables (z)*/

                    /* mise a jour de "etape"                              */
cmac->etape.s1 = s1 ;
cmac->etape.s2 = s2 ;
cmac->etape.outobt = p ;
```

Figure 2. The S→M→A→p Mapping. The C language represents the integral as a repetition of "+=" operators. p is the output of the CMAC; S_1 and S_2 are the inputs; Q is the quantization level. DIMTAB represents the size of the weight tables. The weight tables are stored as an array of Q planes each made of DIMTAB * DIMTAB elements.

and M→A mappings are totally determined by the subroutine's parameters whereas A→p is mainly determined by the input data. The other alternatives would have involved specifying S→M and M→A as binary connection matrices, at a greater expense of computer memory but without significant gains in generality.

CYCLCMAC.C

This subroutine simulates the S→M mapping by executable code as an address translation mechanism. The M→A mapping is handled in a similar fashion, generating the selection to sections of the weight tables that are summed to implement the A→p mapping (Fig. 3).

From outside, the CMAC can be viewed as an array of numbers that are accessible by merely referencing a symbolic name; from the inside it is little more than an address translation and a summation loop, to read, and assignment loop to write or update.

It should be noted, at this point, that even with the overhead of the program and its constants, the CMAC provides a tremendous level of data compression. In fact, with an addressable range of 100 by 100, and 20 quantization levels, the CMAC weight tables occupy only 720 memory locations: better than 10:1 volume reduction. Obviously, there is a certain amount of loss in the process.

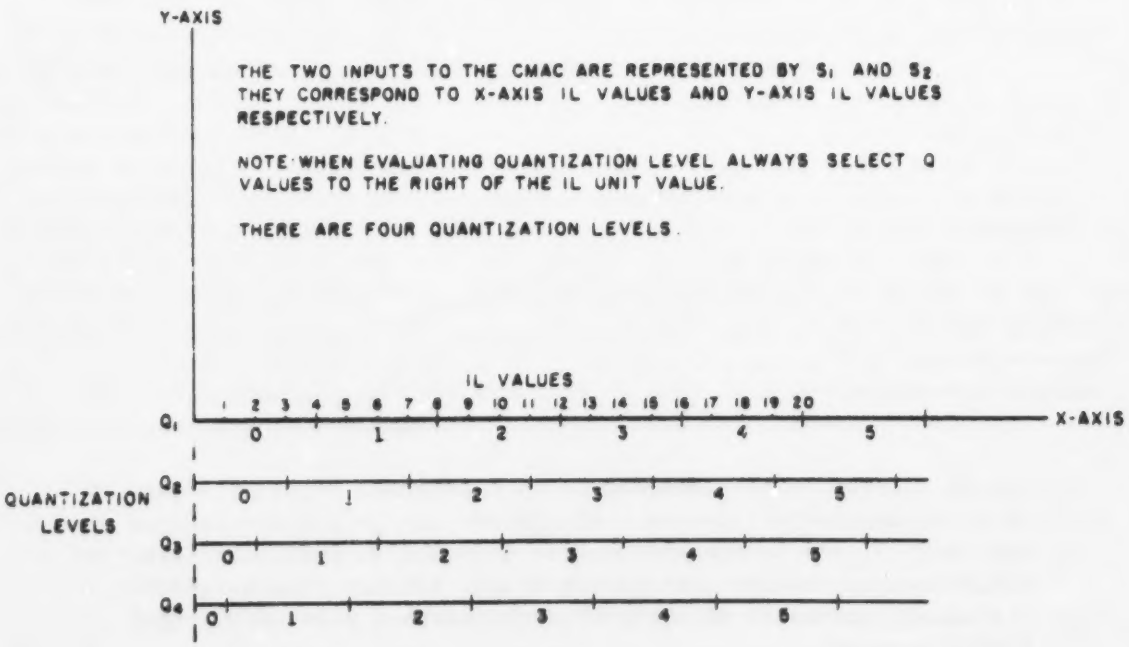
Y-AXIS
THE TWO INPUTS TO THE CMAC ARE REPRESENTED BY S_1 AND S_2
THEY CORRESPOND TO X-AXIS IL VALUES AND Y-AXIS IL VALUES
RESPECTIVELY.
NOTE: WHEN EVALUATING QUANTIZATION LEVEL ALWAYS SELECT Q
VALUES TO THE RIGHT OF THE IL UNIT VALUE.
THERE ARE FOUR QUANTIZATION LEVELS.
IL VALUES
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
X-AXIS
QUANTIZATION LEVELS
Q_1
Q_2
Q_3
Q_4
0 1 2 3 4 5

Figure 3. The evaluation of quantization levels

THE EXPERIMENTS

The experiment consists of the activation of a model containing two CMACs (Fig. 4). In all the experiments the weight tables were initialized at zero. The two CMACs are coupled and have their inputs fed from the output. We have used the model to store coordinate pairs to simulate path following; the plotted data illustrate well the properties under investigation.

In each of the reported experiments, a number of parameters were gradually changed to illustrate their effect on generalization and interference. Each graph also contains the original data, unaltered.

We will describe, successively, the effect of the number of learning cycles on retained data, the number of learning cycles on generalization (tracking effect), the effect of the gain factor, and the learning of complex paths.

In each of these experiments, the model is taught a set of coordinate pairs at addresses corresponding to the previous coordinate pair, in a linked list fashion:

(p1) → p2
(p2) → p3
(pn) → pn + 1

The teaching of the CMAC must be repeated a number of times if an accurate output is desired; this accurate value is henceforth referred to as saturation. It is comforting to see that the saturation level is dependant on the gain factor of the CMAC.

In a "playback" phase, the output of the CMAC is returned to the input, and a starting point is given. The sequence of output values are then plotted. It should be noted that since the inputs are integers and the outputs are real, some truncation occurs during the feedback.

Note that in all the figures referring to paths, the actual shape specifically learned is represented by the solid line and in the diagrams is identified by the letter "A".

To implement this learned path a program called MANAGER is used with the aid of the CMAC algorithm. The MANAGER program requests some starting input vector values.

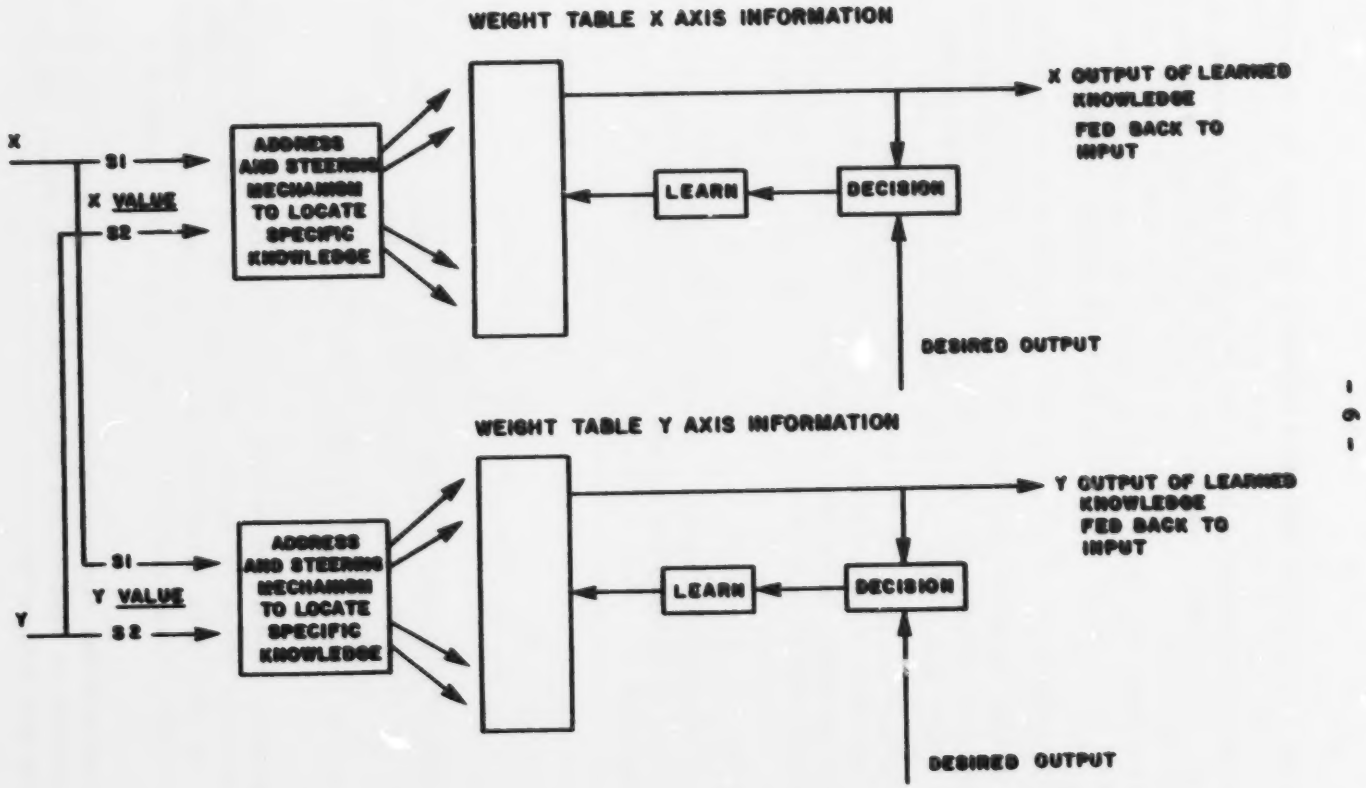

Figure 4. Dual CMAC arrangement used in the experiments

Experiment 1

The following set of experiments illustrates important CMAC properties. A simple path was taught to the CMACs. This involved the teaching of a set of two-dimensional coordinate pairs to a pair of CMACs, the X coordinate to one CMAC and the Y coordinate to the second CMAC cell.

The coordinates used in Experiment 1 are

X	Y
1	0
2	1
3	3
5	5
6	8
8	7
11	9
15	10

The (X, Y) coordinate information (knowledge) is taught to CMAC(X) and CMAC(Y) systems in the following manner. The first set of coordinates i.e., input vectors, (1, 0) points to learned knowledge (2, 1), which is the next pair of coordinates in the wave shape. Coordinate pair (2, 1) is used as input vector to point to learned knowledge (3, 3). This ordered sequence of information storage continues to produce a picture of the path, within the two CMAC cells.

This basic demonstration illustrates what happens when a sequence is saturated, and playback is initiated outside of the learned curve. The curve B bears obvious similarities with curve A.

Input S_1 and S_2		Actual X output*	Actual Y output*
x	y		
3	2	4.99	3.99
7	9	9.49	7.02
10 -	7	14.99	7.026

*These columns refer to line B.

Observe the similarity between lines A and B on Fig. 5.

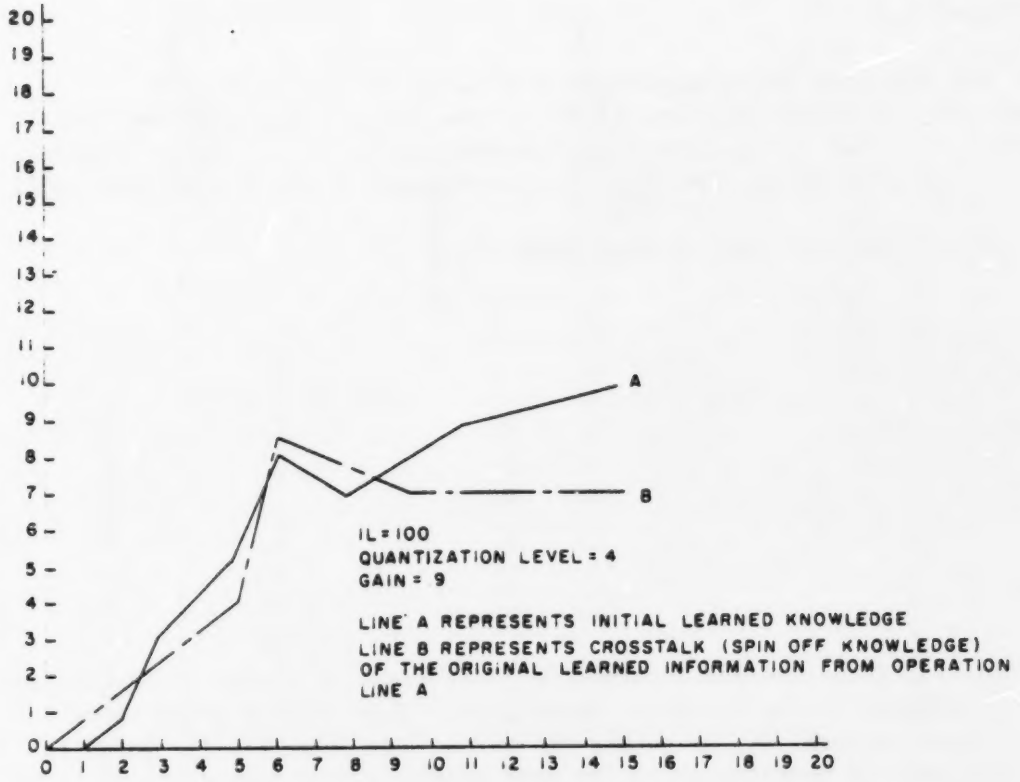

Figure 5. Learning of information (simple waveform)

During the learning of line A, that information had been stored in a set of weight cells. A proportion of those weight cells are shared by line B input vectors. Consequently the knowledge that line B illustrates is a generalization from the original learned waveform.

The deflection of line B, as it nears the end of line A, occurs because the information learned near the end of the line A interferes less with surrounding weight cells. This is because nothing is learned beyond this point.

Certain conclusions can be drawn from Experiment 1:

(1) The learning of this simple waveform illustrates the generalization effect: similar starting points produce similar paths.

(2) This differences between original path and playback path are commensurate with the distance of their origins.

(3) In summary the generalization occurs because each coordinate shares a portion of the cells it addresses with its neighbours.

Experiment 2

In this experiment a more difficult shape was learned. The experiment had two purposes: (A) to investigate how the number of learning operations affect the complete learning of a shape; (B) to investigate the tracking effect that occurs when input vectors given are other than those previously learned for that particular path.

Figures 6 to 8 illustrate Experiment 2. In computing each of three diagrams the CMAC parameters used were IL = 100, Quantization level = 20, Gain = 0.9.

The following is a list of X, Y coordinates learned for Experiment 2.

X	Y
10	10
20	30
30	50
50	60
70	70

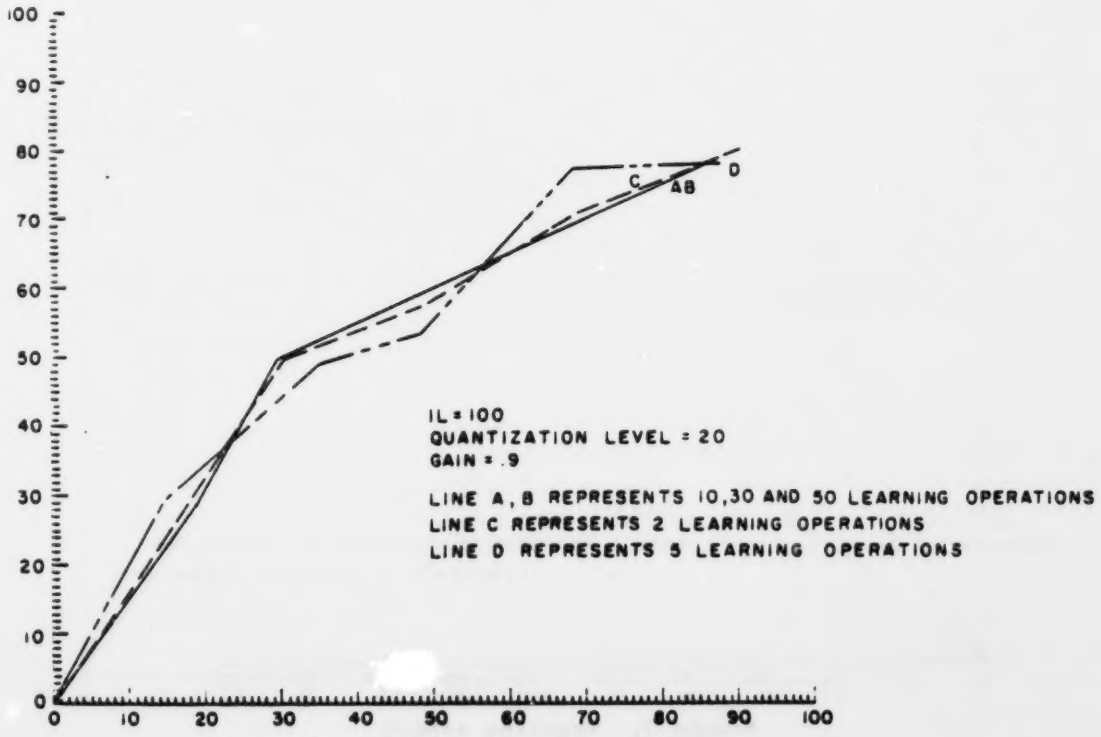

Figure 6. Experiment 2

Observations

There are four paths represented in Fig. 6: A, B, C, and D. Lines A and B are on top of one another; they represent 50 and 10 learning operations, respectively, of the above coordinate table. Lines C and D represent five and two learning operations, respectively, of the above coordinate table.

Similar to previous experiments, each of the paths were generated by inputting the initial vector (10, 10) into a CMAC weight table and learning the coordinate table a specified number of times. The resulting output knowledge (coordinates) from input (10, 10) are plotted, and are used as pointers to the next set of coordinates.

This part of the experiment (shown in Fig. 7) examines the tracking effect that is the result of the generalizing effect.

Line A represents the CMAC weight table that has 50 learning operations. Line B represents 10 learning operations of the same table.

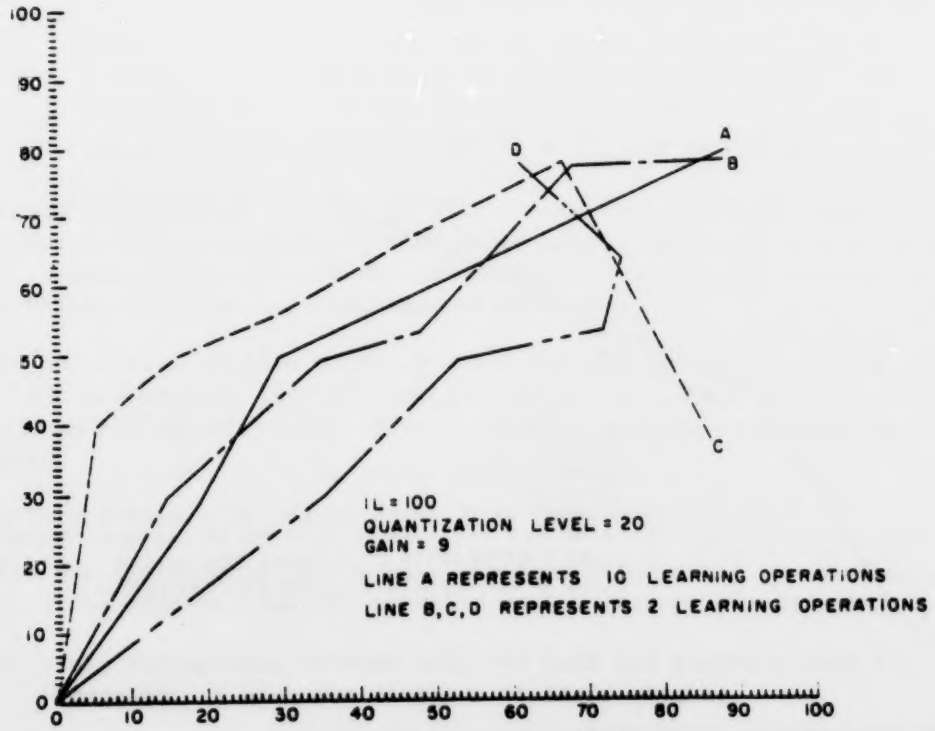

Figure 7. Tracking effect

Lines C and D were computed from the CMAC that had two learning operations performed. Line C was computed by giving input vectors X = 0 and Y = 20. The resulting output was plotted. In the same manner line D was computed using input vectors X = 20 and Y = 10.

Lines C and D follow or track line A or B. None of the initial coordinates given for lines C and D were selected from the original path learned. Some of the cell weights (locations) that are associated with those input vectors of lines C and D are also shared by original learned vector inputs. It is their influence on the cells that produces the paths such as lines C and D.

Lines C and D deflect abruptly at the end of line A. The lack of data interrupts the blending of weighted values in the CMAC cell that describe a particular point about the path.

To achieve the results illustrated in Fig. 8, the CMAC parameters were unchanged, at IL = 100, Q = 20, Gain = 0.9. The coordinate table used was identical to the previous parts of Experiment 2.

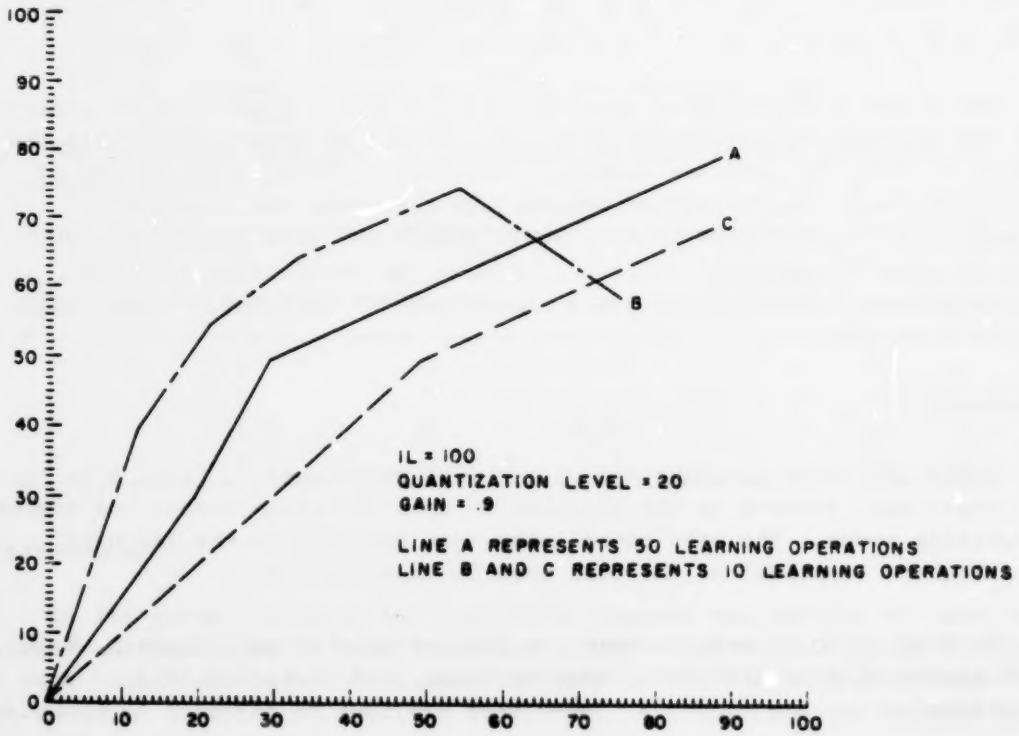

Figure 8. Experiment 2

The same initial input vector was given as in part 2 of Experiment 2. Line B was started with X = 0 and Y = 20 which corresponds to Fig. 7, line C. Line C was started with X = 20 and Y = 10, similar to Fig. 7, line D.

The stored knowledge relating to these paths in the separate CMAC weight tables was influenced by two learning operations in Experiment 2, part 2, and 10 learning operations in Experiment 2, part 3.

A comparison can be made between tracking lines of parts 2 and 3 of Experiment 2. Line A (50 learning operations) is selected as the normal in both parts (Figs. 7 and 8), and one can see how lines B and C (10 learning operations) in part 3 take a more evenly distributed course. Lines B and C of part 3 relate directly to a greater number of learning operations. It is interesting to note that in Experiment 2, part 2, with only two learning operations or 4% of the normal line learning operations, there is such tracking ability.

The effect of the diminishing information again causes line B to abruptly change course. Line C in this experiment does follow line A a little further. This could not be repeated with other vectors and is rather fortuitous. The next move (not shown) for line C is a track that leads back to zero.

The tracking experiments were carried out with a control as to when to stop the tracking path. In the experiment shown, an intersection with the normal, or when the path was within a specified distance from the end of the normal, were the controlled points. In all cases the tracking, if allowed to go beyond these points, would within two move operations back-track to zero (coordinate) position, because the coordinates at these control points would point to areas in CMAC memory that had no previously learned knowledge.

Experiment 3

There are three diagrams associated with Experiment 3, Figs. 9 to 11. This experiment focuses on the relation of gain variation versus the number of learning loops. The path coordinates that are used in the learning operation are identical to the ones used in Experiment 3.

In Figs. 9 to 11 each diagram has four paths plotted. Lines A, B, C, and D represent four different learning gains, 0.9, 0.6, 0.4, 0.2, respectively.

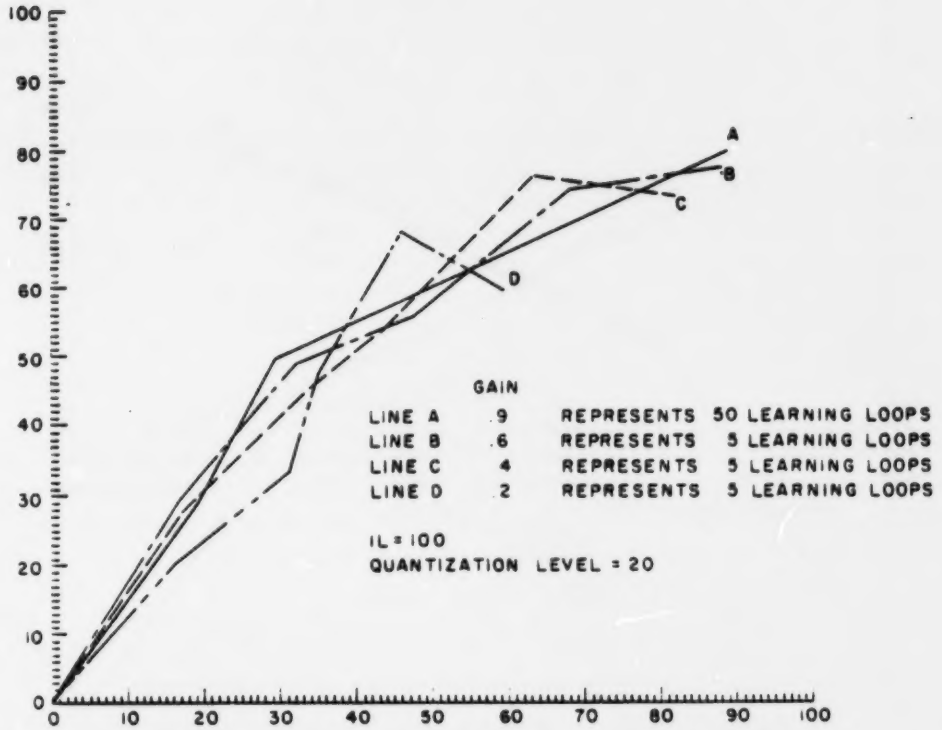

Figure 9. Gain vs. number of learning loops

The following is a list of parameters associated with each part (1, 2, 3) in Experiment 3.

Part 1			Part 2			Part 3		
Line	Loop	Gain	Line	Loop	Gain	Line	Loop	Gain
A	50	0.9	A	50	0.9	A	50	0
B	5	0.6	B	10	0.6	B	30	0
C	5	0.4	C	10	0.4	C	30	0
D	5	0.2	D	10	0.2	D	30	0.2

Observations

In all parts, lowering the gain value lessens the ability to learn as quickly the knowledge at hand, compared to the other higher levels of gain in each individual part. Note that the gain is the factor that relates to the speed of learning of specific knowledge. A comparison can be made of

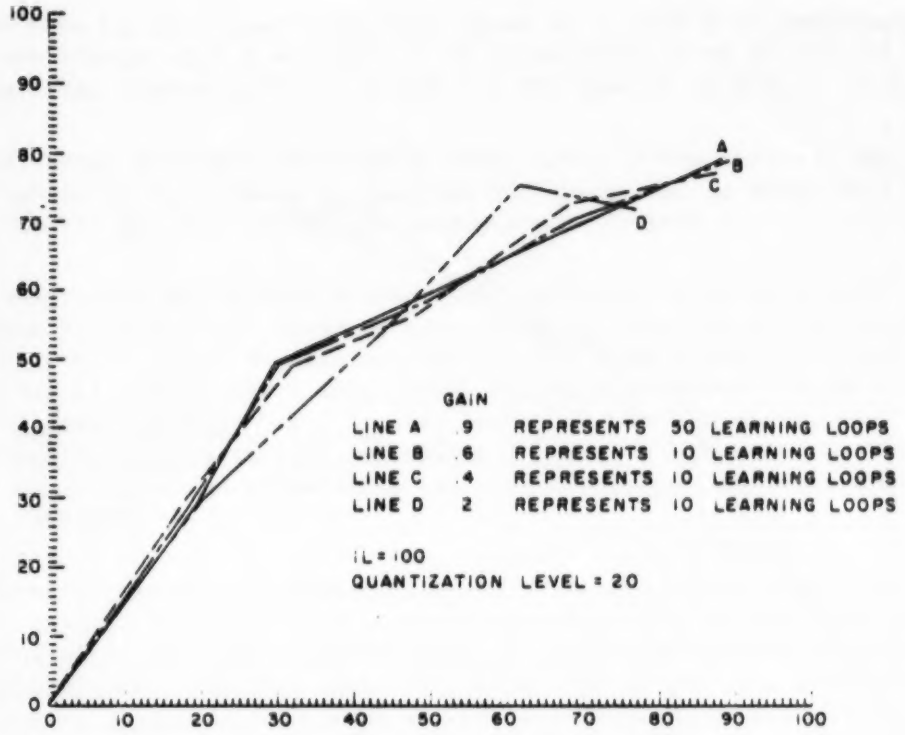

Figure 10. Gain vs. number of learning loops

the increase in learning loops between each part and how the rate of knowledge learned increases, at the lower gain values. In part 3 the 30 learning loop system has a saturation effect on the learning of the specific knowledge. The lower gain values reflect very little influence on the rate of the learning procedure.

In conclusion, Experiment 3 confirms that the CMAC parameters Gain and number of learning loops can be used as a fine tuning mechanism, which would provide the means of operating a series of CMACs in an efficient manner with regard to computer execution time.

Experiment 4

This experiment is a demonstration of generalisation that utilises a more complex path, which forms a closed loop structure.

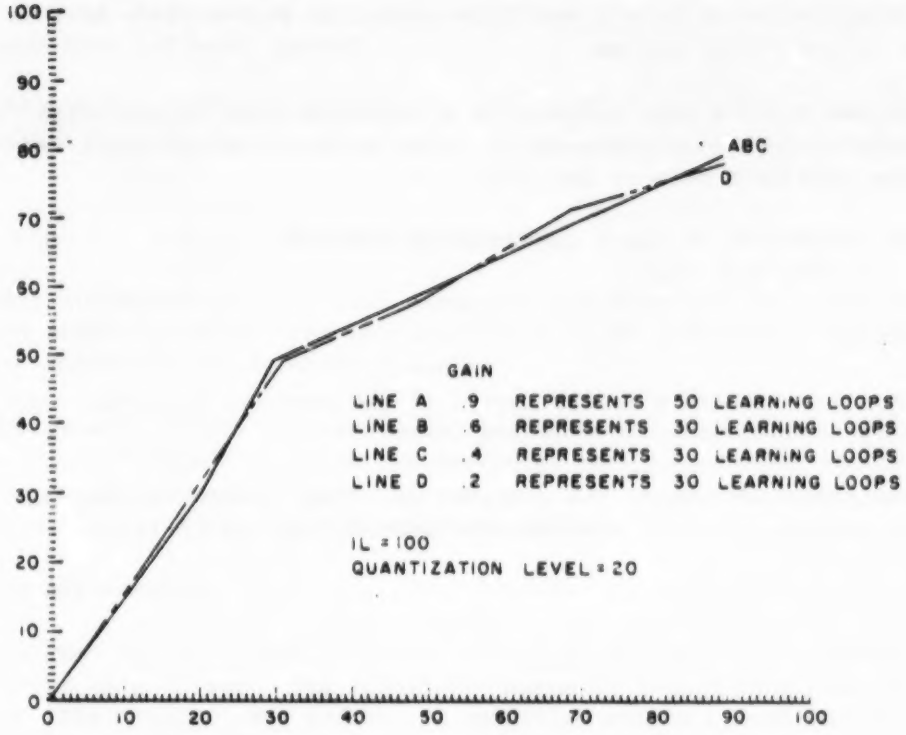

Figure 11. Gain vs. number of learning loops

The waveshape to be learned had the following set of coordinates:

X	Y
3	3
3	5
1	7
2	9
4	10
6	9
7	10
8	6
7	4
2	2

The inputs for the CMAC representing X would be $S_1 = 3$ and $S_2 = 3$. They would be provided with the preceding X coordinate "3", which will be deposited in the CMAC memory.

The same procedure would follow for "Y" CMAC representing Y coordinates: $S_1 = 3$, $S_2 = 3$ and 5 the preceding Y coordinate that is to be learned by the Y CMAC system.

The two X and Y CMAC systems, in a manner similar to preceding experiments, use a loop structure to learn portions of the specified knowledge upon each pass of the loop.

The parameters of the X and Y CMAC system are: Gain = 0.9, IL = 100, and Q = 20 (See Fig. 12.)

Observations

The trace labeled A is the result of the expected structure that was explicitly learned using 50 learning loops.

When this information was recalled from CMAC memory the initial starting position for plotting of this information was (2, 2).

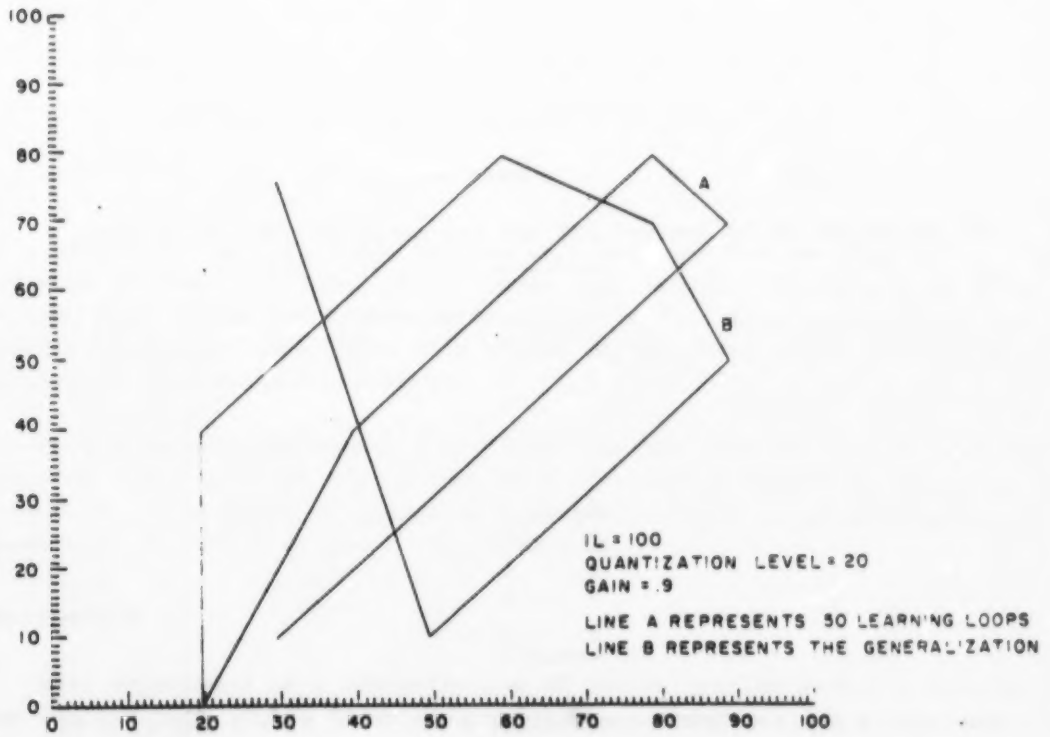

Figure 12. Learning of a complex path

The other paths labeled B, C, and D were initiated by requesting, from X and Y CMACs, information that did not initially follow the explicit knowledge that had been learned.

Observe the different starting points from (2, 2) and how the trend of information quickly aligns itself with the originally learned information.

GENERAL ASSESSMENT

This investigation into CMAC structure and operation has revealed some exciting characteristics that one can relate to as a simulation of human learning mechanism at an "atomic" scale.

The generalization phenomenon can easily be paralleled with its biological counterpart in which new knowledge influences other knowledge areas to create a dynamic memory structure. This effect is most interesting in simulating human learning characteristics.

Possible Applications

The CMAC can be viewed, in turn, as a look-up table, as a control element, or as a filter. Its simplicity makes it a good candidate for hardware integration. The addressing mechanism offers a powerful data compression method (≈10:1) and the algorithm is very easy to use as an operator in a parallel processor.

The convergence it exhibits in path following experiments make it a likely candidate as a control element for an auto-adaptive system.

Furthermore, the CMAC has good noise and error immunity that are desirable in fail-safe systems.

Areas for Future Investigation

One of the largest costs in the implementation of any learning system is that of the learning phase itself. The CMAC is an ideal workbench for investigating a mechanism of specifying and stabilizing knowledge. We are exploring an estimator of pertinence that could be used to channel information to an appropriate section of a learning system.

This estimator would provide a confidence level to accept or reject new data as they are fed to the CMAC.

ACKNOWLEDGEMENTS

We want to express our deep gratitude to Mr. Timothy Booth who played a major role in carrying out this research, implementing the algorithms in this report on a real machine producing real results; he was also the creator of many programs essential to presenting the data in graphics and performing the actual simulation runs.

REFERENCES

1. Albus, L.S. Brains, Behavior, and Robotics. Byte Publications Inc. 1981.

2. Minsky, M., and Papert, S. Perceptrons. The Massachusetts Institute of Technology. 1969.

APPENDIX A

/* stdeclar.c */

```
/* Ce fichier contient les declarations communes a tous les programmes.   */
/* Il doit etre inclus au debut de chaque fichier a l'aide de la commande */
/* #include "stdeclar.c" . Toute modification a ce fichier rend obliga-   */
/* toire une recompilation de tous les programmes y referant.             */

/* Diane Goupil, JUILLET 84                                               */

#include <stdio.h>
                        /* definit les routines d'entree sortie standard, */
                        /* et entre autres, NULL = 0 .                    */
#include <math.h>
                        /* librairie mathematique, contient ceil, exp,    */
                        /* log, etc.                                      */

#define OK      1
                        /* valeur retournee par certaines fonctions si-   */
                        /* gnifiant un deroulement normal.                */
#define LOG2    0.6931471805599453
                        /* logarithme naturel de 2, sert a la conversion  */
                        /* de log. naturel vers log. base 2 .             */

struct streta
                        /* structure contenant l'information concernant   */
                        /* une etape d'apprentissage/consultation         */
   {
   int s1 ;             /* premier stimulus ( en x )                       */
   int s2 ;             /* deuxieme stimulus ( en y )                      */
   float outesp ;       /* sortie esperee                                  */
   float outobt ;       /* sortie obtenue                                  */
   } ;

typedef struct streta typetape ;
                        /* type definissant la structure d'une etape       */

struct strcmac
                        /* structure contenant toutes l'information        */
                        /* propre a un cmac.                               */
   {
   char *nom ;          /* chaine contenant le nom du cmac                 */
   int no ;             /* numero du cmac                                  */
   int dimcma ;         /* dimension de la plage adressable du cmac        */
```

```
    int q ;                 /* niveau de quantification (ou nombre de tables)  *
    float gain ;            /* facteur de gain associe a ce cmac               *
    int dimtab ;            /* dimension des tables du cmac                    *
    int nbelecma ;          /* nombre total d'elements dans le cmac            *
    int nbeletab ;          /* nombre total d'elements dans les tables         *
    float *tabpon ;         /* pointeur sur l'espace alloue pour les tables    *
                            /* l'espace pour les tables est alloue dynamique-  *
                            /* ment en fonction de nbeletab. Il s'agit d'un    *
                            /* long vecteur qu'on peut manipuler comme un ta-  *
                            /* bleau tridimensionnel en calculant l'indice     *
                            /* en consequence. L'indice qui parcourt les ta-   *
                            /* bles (z) varie le plus vite, suivi de celui qui *
                            /* parcourt s2 (y), puis s1 (x).                   *

    typetape etape ;
                            /* structure contenant les variable relative a la  *
                            /* derniere etape d'apprentissage/consultation     *
    struct strcmac *suivant ;
                            /* pointeur sur le cmac suivant dans la liste      *
    } ;

typedef struct strcmac typcmac ;
                            /* type definissant la structure d'un cmac         *

typedef typcmac *typtcmac ;
                            /* type definissant un pointeur sur un cmac        *

                                /* fin de stdeclar.c */
```

/* consulte.c*/

```
/* consulte calcule et retourne pour les coordonnees fournies en entree la  */
/* valeur de sortie pour le cmac. La structure "etape" du cmac est aussi     */
/* mise a jour.                                                              */
/* La sortie est la somme des q valeurs contenues dans les q tables de       */
/* ponderation a la case specifiee par (s1+i)/q, (s2+i)/q, i, ( i=1,q )      */

/* version inspiree de cmacx.c de T. Booth                                   */
/* Diane Goupil , 29 JUIN 84                                                 */

#include "stdeclar.c"
                    /* declarations standard                                 */

float consulte( cmac, s1, s2 )

typtcmac cmac ;     /* pointeur sur la structure de cmac                     */
int s1,s2 ;         /* entree en x et y ( stimulus 1 et 2 )                  */
{

float p;            /* valeur cumulee pour le resultat */
int i;

p = 0.0;


for (i = 0; i < cmac->q; ++i)
                    /* on utilise tabpon comme un tableau tridimensionnel    */
   p += cmac->tabpon [
      (( s1 + i ) / cmac->q ) * cmac->dimtab * cmac->q +   /* indice s1 (x) */
      (( s2 + i ) / cmac->q ) * cmac->q +                  /* indice s2 (y) */
      i ] ;                                              /* indice tables (z) */

                    /* mise a jour de "etape"                                */
cmac->etape.s1 = s1 ;
cmac->etape.s2 = s2 ;
cmac->etape.outobt = p ;

return(p);

}                    /* fin de consulte */
```

/* cyclcmac.c */

```
/* cyclcmac calcule et retourne pour les coordonnees fournies en entree la  */
/* valeur de sortie pour le cmac. Selon la valeur de learn, un apprentis-   */
/* sage peut avoir lieu. L'apprentissage s'effectue ainsi : la difference   */
/* est calculee entre la valeur esperee et la valeur obtenue pour la sortie */
/* puis on la multiplie par le facteur de gain et on la divise par le ni-   */
/* veau de quantification ( ou nombre de tables ) et on ajoute la valeur    */
/* ainsi calculee aux positions appropriees dans les tables de ponderation  */
/* ( specifiees par (s1+i)/q, (s2+i)/q, i, pour i=1,q )                     */
/* La structure "etape" du cmac est mise a jour.                            */

/* version inspiree de learnx.c de T. Booth                                 */
/* Diane Goupil , 16 JUILLET 84                                             */

#include "stdeclar.c"
                    /* declarations standard                                */

float cyclcmac( cmac, s1, s2, learn, outesp )

typtcmac cmac ;     /* pointeur sur la structure de cmac                    */
int s1,s2 ;         /* entree en x et y ( stimulus 1 et 2 )                 */
int learn ;         /* indique si un apprentissage va avoir lieu :          */
                    /*    learn = NULL  -> pas d'apprentissage              */
                    /*    learn = OK    -> apprentisage desire              */
float outesp ;      /* valeur esperee de la sortie                          */

{

int i;
float delta ;       /* correction a ajouter aux tables pour l'apprentissage */

consulte (cmac, s1, s2, learn, outesp );

if ( learn != NULL )
   {
   delta = (( cmac->gain * ( outesp - cmac->etape.outobt )) / cmac->q ) ;
   for (i = 0; i < cmac->q; ++i)
                    /* on utilise tabpon comme un tableau tridimensionnel   */
      cmac->tabpon [
      (( s1 + i ) / cmac->q ) * cmac->dimtab * cmac->q +   /* indice s1 (x) */
      (( s2 + i ) / cmac->q ) * cmac->q +                  /* indice s2 (y) */
      i ] += delta ;                                  /* indice tables (z) */
```

```
                    /* mise a jour de "etape"                                    */
    cmac->etape.outesp = outesp ;
    }

return( cmac->etape.outobt ) ;

}                                   /* fin de cyclcmac */
```